Express Mail No. EL777309296US
Docket No. SJO920000171US1
Firm No. 0037.0011

METHOD, SYSTEM, AND PROGRAM FOR MANAGING CLIENT ACCESS TO A SHARED RESOURCE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a method, system, and program for managing client access to a shared resource.

2. Description of the Related Art

A storage manager software product, such as the International Business Machines Corporation ("IBM") Tivoli Storage Manager, manages the backup and archiving of client application data at a central server. Multiple client computers may provide data to a server including the IBM Adstar Distributed Storage System (ADSM) or the Tivoli Storage Manager, to backup on a storage array, such as numerous hard disk drives or tape medium. Upon receiving the data, the storage manager product typically writes the data updates to a log file. If the system goes down, then any changes to the database can be reversed using the updates maintained in the log file to ensure that the system is consistent as of a specified point in time prior to the system failure.

After a client has transferred a certain amount of data, the storage manager then commits the client transactions to the storage device. Committed transactions can then be removed from the log file. After commitment, a client is ensured that the storage device is consistent with respect to the client as of the time of the most recent committed transaction. For instance, when a Tivoli Storage Manager client is in session with a Tivoli Storage Manager server and there is an interruption of any kind, the Tivoli Storage Manager server automatically rolls the session back to the last committed transaction.

In the current art, if there are numerous clients providing backup data to the storage manager, then the log file may become full. When the log file becomes full, the storage manager

must take the server off-line to prevent any further backup operations because the log file cannot be used to store the updates for data recovery purposes in the event of a system failure.

For these reasons, there is a need in the art for managing access to network resources, such as a log file, in systems where multiple clients are competing for shared resources. Further, there is a need to manage the log file to prevent the log file from becoming full, which takes the server off-line.

SUMMARY OF THE PREFERRED EMBODIMENTS

Provided is a method, system, and program for managing client transactions requesting access to a shared resource. Client transactions are logged in a log file from multiple clients. A determination is made of one client transmitting data at a transmission rate less than a threshold transmission rate. Subsequent transactions from the determined client accessing the shared resource are denied to provide additional space in the log file for new transactions from additional clients requesting access to the resource.

In further implementations, all pending transaction of the determined client are removed from the log file.

Yet further, clients submit transactions requesting the resource during a session that the clients initiate. A determination is made of one client session active longer than a threshold time period. The determination of whether the client data transmission rate is less than the threshold transmission rate is made for the determined client whose session is active longer than the threshold time period. Subsequent transactions requesting access to the resource are denied for the client having the session active longer than the threshold period of time and having the data transmission rate less than the threshold transmission rate.

In certain implementations, an oldest pending transaction logged in the log file is capable of preventing new transactions from being added to the log file.

Yet further, the resource may comprise a storage device and the transactions may provide updates to data in the storage device.

The described implementations provide a technique for managing access to a shared resource that removes pending transactions from a log submitted by clients that are transmitting data at an unacceptably low data transmission rate. This methodology prevents clients transmitting at particularly low data transmission rates from acting as a bottleneck that prevents the submission of new transactions from clients transmitting at higher data transmission rates. In this way, the described implementations provide a technique for performance tuning at a server providing access to a shared resource by dynamically managing active clients and, in particular, removing transactions from those clients transmitting at relatively slow data transmission rates.

BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 is a block diagram illustrates a network storage environment implementation of the present invention;

FIG. 2 illustrates an implementation of a log file used to store transactions providing updates to the storage device; and

FIG. 3 illustrates logic implemented in a storage manager to manage client access to the shared resource through the log file in a manner that improves system performance.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In the following description, reference is made to the accompanying drawings which form a part hereof and which illustrate several embodiments of the present invention. It is understood that other embodiments may be utilized and structural and operational changes may be made without departing from the scope of the present invention.

FIG. 1 illustrates a network storage system implementation of the invention. A server 2 controls input/output (I/O) access to a storage device 4, such as an array of hard disk drives

(e.g., Direct Access Storage Device (DASD), Just a Bunch of Disks (JBOD), Redundant Array of Inexpensive Disk (RAID) array, etc.), a tape subsystem, optical storage system, etc. Numerous clients 6a, b, c communicate with the server 2 over network 8. The server 2 may comprise any server class machine or other computer system capable of managing access to a storage device. The clients 6a, b, c may comprise any computer device known in the art, such as a server, mainframe, desktop computer, network client, laptop, etc. The network 8 may comprise any network system known in the art, such as a Local Area Network (LAN), Storage Area Network (SAN), Intranet, the Internet, etc., using any communication protocol known in the art, e.g., the Transmission Control Protocol/Internet Protocol (TCP/IP), Fibre Channel, Small Computer System Interface (SCSI), etc.

In certain implementations, the server 2 includes a storage manager 10 program to manage I/O requests from the multiple clients 6a, b, c. The clients may include a client backup application program to backup data at the storage device 4 through the server 2. In the described implementations, the storage manager 10 maintains a log file 12 of update transactions received from the clients 6a, b, c. After a client has transferred a predetermined number of transaction bytes or a predetermined number of files, the update transactions for that client 6a, b, c are applied to the storage device 4 and then marked as committed in the log file 12. Committed transactions are removed from the log file 12 to free log file space 12.

FIG. 2 illustrates an implementation of how the log file 12 stores transactions in the log file 12 in a manner known in the art. In FIG. 2, the log file 12 is comprised of sequential blocks of data, such as sequential fixed block addresses (FBA). The storage manager 4 maintains a tail pointer 20 and pin pointer 22 to point to block addresses in the log file 12 including client transactions. The tail pointer 20 points to the most recent added transaction, which in FIG. 2 is transaction 4. A new transaction is added to the block immediately following the tail pointer 20 and the tail pointer 20 is then moved downward to point to the next block address including the added transaction. The pin pointer 22 points to the oldest transaction in the log file 12. If the tail pointer 20 reaches the last block address in the log file 12, then the tail pointer wraps

around to the first block address in the log file 12. After wrapping around, the tail pointer 12 would continue to move downward from the first block address as transactions are added. However, the tail pointer 20 cannot move past the pin pointer 22. The log file 12 is full and unable to accept further update transactions if the tail pointer 20 is at a block address adjacent to the block address pointed to by the pin pointer 22. When the transaction addressed by the pin pointer 22 is committed, the pin pointer 22 is moved downward to the next oldest transaction maintained in the log file 12.

With the log file 12 arrangement illustrated in FIG. 2, a client 6a, b, c transmitting backup updates at a relatively slow data transmission rate may not transmit enough data to trigger a commit operation. In fact, the pin pointer 22 may point to the oldest transaction from the client 6a, b, c transmitting data for a long time at a particularly slow rate without triggering the commit procedure. In such case, the pin pointer 22 may become a bottleneck that prevents clients 6a, b, c transmitting backup updates at particularly high data transmission rates from providing their updates because new update transactions cannot be added beyond the pin pointer 22 addressing the old pending transaction from the slow client 6a, b, c. Moreover, even if transactions between the tail 20 and pin 22 pointers are committed and removed from the log file 12, such free space cannot be used until the transaction addressed by the pin pointer 22 is committed. Upon commitment, the pin pointer 22 may advance beyond the free space to make such space available to new transactions.

The described implementations provide a methodology to manage existing client 6a, b, c sessions to prevent bottlenecks resulting from clients 6a, b, c transmitting at particularly slow transmission rates. A client 6a, b, c may transmit at a relatively slow rate if it uses a slower interface adaptor, e.g., a telephone modem, when other clients are using fast connections, e.g., optical networks. Clients 6a, b, c that are overloaded may also transmit data at slow rates.

FIG. 3 illustrates logic implemented in the storage manager 4 to manage client 6a, b, c sessions to prevent sessions from becoming bottlenecks for the other managed sessions. In the implementation of FIG. 3, the storage manager 4 considers a time threshold and throughput

threshold to determine whether to terminate a client session. Control begins at block 100 with the storage manager 4 initiating a performance tuning operation, that may be initiated at periodic intervals. The storage manager 4 performs the loop at block 102 through for each client 6a, b, c session *i*. The storage manager 4 may consider client sessions in the order of the time they have been active, or any other parameters. If (at block 104) the time client session *i* has been idle, i.e., not transmitted data, exceeds the idle threshold, then the storage manager 4 terminates (at block 106) that idle client session *i*. The storage manager 4 further reverses (at block 108) any uncommitted updates partially applied to the storage device 4 and removes the uncommitted update transactions from the log file 12. In certain implementations, whenever reversing or removing uncommitted update transactions from the log file 12, the storage manager 4 "rolls back" or removes the updated data maintained in the reversed or aborted transaction from the storage device 4 so that the data in the storage device 4 remains consistent. For instance, updates from uncommitted transactions can be reversed from the storage device 4 to make the data in the storage device 4 consistent as of the time of the last committed update.

If (at block 110) one of the uncommitted transactions removed from the log file 12 was addressed by the pin pointer 22, then the storage manager 4 advances (at block 112) the pin pointer 22 to the next oldest uncommitted transaction in the log file 12. If the pin pointer 22 did not address one of the removed uncommitted transactions (from block 110) or after advancing the pin pointer 22 (at block 112), control proceeds to block 120 to consider the next ($i + 1$)th client 6a, b, c session until all client 6a, b, c sessions are considered.

If (at block 104) the client session *i* does not exceed the idle threshold, then the client determines (at block 114) whether client session *i* has been active for a time exceeding the time threshold, e.g., a few hours or any predetermined time. If so, the storage manager 4 determines (at block 116) the data throughput rate of client session *i*. In certain implementations, the storage manager 4 may calculate the data throughput rate as the number of bytes transferred since the start of the client 6a, b, c session divided by the time the client session 6a, b, c has been active. Alternative techniques may be used to calculate the throughput, such as

consideration of the throughput of only more recent data transmissions. If (at block 118) the determined throughput is less than the throughput threshold, then control proceeds to block 106 to terminate client session *i* and advance the pin pointer 22 if the oldest uncommitted transaction was from client session *i*. Otherwise, if the throughput threshold is exceeded, then control proceeds (at block 120) to consider the next client session *i*.

The time, throughput, and idle thresholds used in the logic of FIG. 3 may be determined based on empirical tests of the storage network system to determine an optimal selection of threshold values. Additionally, the time, throughput, and idle thresholds may be adjusted using line commands from the server 2 or from a client 6a, b, c over the network 8.

The performance tuning methodology of FIG. 3 provides for dynamic management of active client sessions to determine whether to terminate a session to free log file resources. The methodology of FIG. 3 tends to terminate those client sessions creating a bottleneck at the pin pointer 22 because client sessions that have been active for a relatively long period of time and that have low data transmission rates are likely to have the oldest uncommitted transactions. Terminating those client 6a, b, c sessions that are most likely to have the oldest uncommitted transactions would advance the pin pointer 22 to make room for more update transactions, in addition to freeing log file 12 space. In this way, a client session with a slow throughput will not prevent client sessions having relatively high throughput rates from continuing operations at the high throughput rate.

The above methodology is particularly advantageous if the storage network has numerous clients transmitting data at significantly different rates. Moreover, if one client is overloaded or having errors, yet still transmitting update transactions, then the methodology of FIG. 3 prevents the problematic client session from affecting all active client sessions.

In the implementation of FIG. 3, a client satisfying the thresholds is terminated. In alternative implemetnations, those clients satisfying the thresholds may be directed toward another server instead of being terminated. For instance, one server can be dedicated to clients 6a, b, c transmitting at relatively slow rates so that clients 6a, b, c having higher throughput can

use the same server and not have their performance adversely affected by slower clients 6a, b, c.

Providing an alternative server for slower data transmissions would be particularly useful for Internet commerce web sites. Those clients accessing the Internet commerce web site at slow transmission rates would access the same server so that clients transmitting at relatively faster rates can use the "fast" server and not be limited by clients with slower throughput.

The following describes some alternative implementations.

The preferred embodiments may be implemented as a method, apparatus or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" as used herein refers to code or logic implemented in hardware logic (e.g., an integrated circuit chip, Field Programmable Gate Array (FPGA), Application Specific Integrated Circuit (ASIC), etc.) or a computer readable medium (e.g., magnetic storage medium (e.g., hard disk drives, floppy disks,, tape, etc.), optical storage (CD-ROMs, optical disks, etc.), volatile and non-volatile memory devices (e.g., EEPROMs, ROMs, PROMs, RAMs, DRAMs, SRAMs, firmware, programmable logic, etc.). Code in the computer readable medium is accessed and executed by a processor. The code in which preferred embodiments are implemented may further be accessible through a transmission media or from a file server over a network. In such cases, the article of manufacture in which the code is implemented may comprise a transmission media, such as a network transmission line, wireless transmission media, signals propagating through space, radio waves, infrared signals, etc. Of course, those skilled in the art will recognize that many modifications may be made to this configuration without departing from the scope of the present invention, and that the article of manufacture may comprise any information bearing medium known in the art.

The described implementations provided a technique for managing access to a storage device. Additionally, the above described logic may be used to manage client access to any

type of shared resource, such as any type of input/output (I/O) device (e.g., printer, scanner, etc.), network adaptors, controllers, etc.

In the described implementations, the log file 12 stored transaction updates in a circular buffer. In alternative implementations, different queuing techniques may be used to store updates in the log file 12.

In the logic of FIG. 3, a client session was deemed too slow if it has been active a relatively long time, i.e., exceeding the time threshold, and has a relatively low throughput rate, i.e., less than the throughput threshold. Additional or different performance parameters may be considered to determine if a client session is transmitting data at a sufficiently slow rate that increases the likelihood that the client session is a bottleneck to faster client sessions.

The preferred logic of FIG. 3 described specific operations occurring in a particular order. In alternative embodiments, certain of the logic operations may be performed in a different order, modified or removed and still implement preferred embodiments of the present invention. Morever, steps may be added to the above described logic and still conform to the preferred embodiments.

Therefore, the foregoing description of the preferred embodiments of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto. The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.